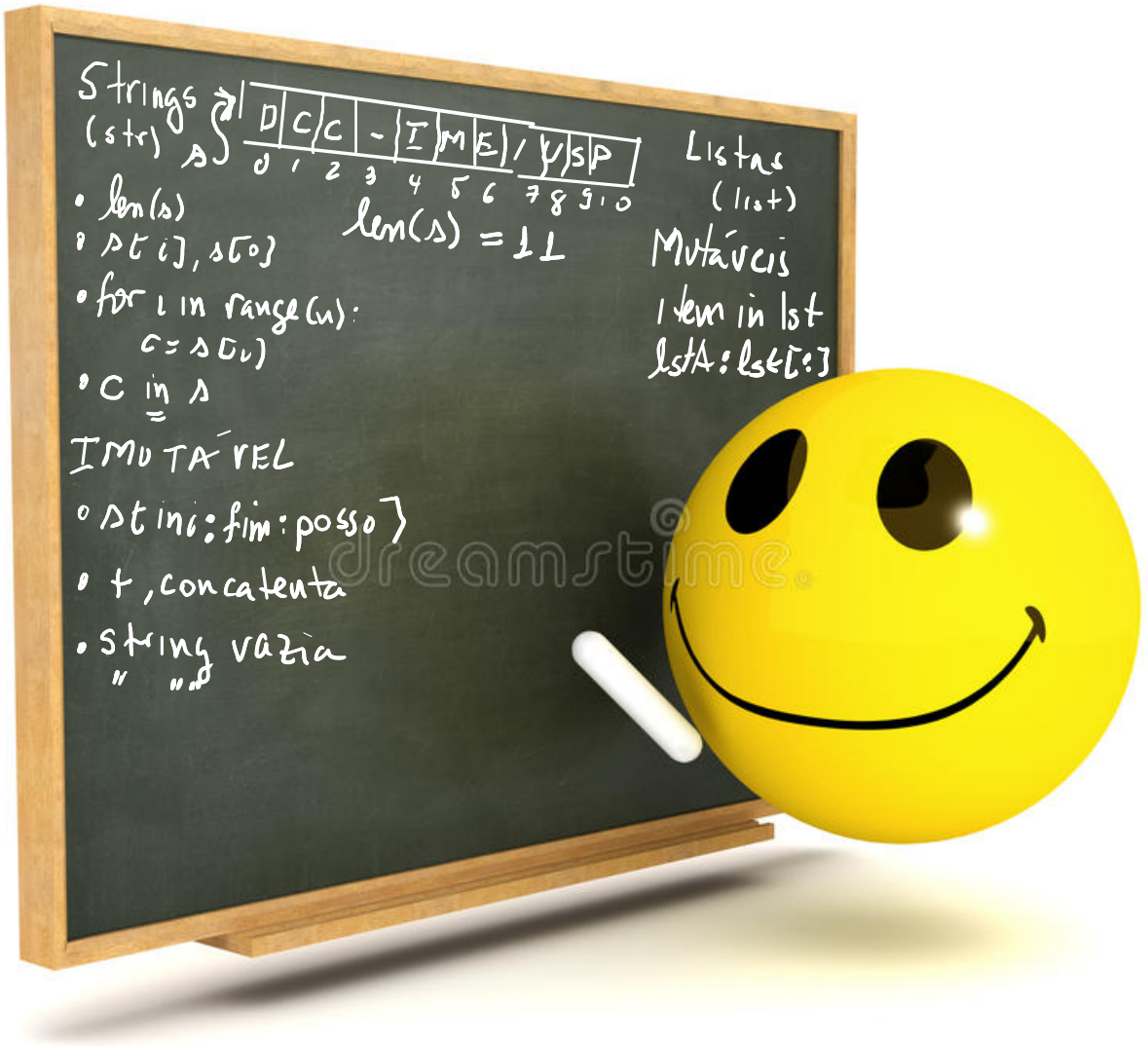
20 Reunião 20: 29/JUN/2021

Figure 1: Mafalda por Quino

20.1 Reuniões passadas

- listas
- criar listas
- acessar itens da lista através de **índices**
- percorrer listas
- função `len()`
- comando de repetição `for ... in range(ini, fim, passo):...`
- função `sum()`
- fatias: `lst[ini: fim: passo]`
- fatias em python
- apelidos × clones
- mutabilidade: **última reunião**
- funções mutadoras: **última reunião**

20.2 Hoje


Strings
(str)
D C C - I M E / V S P
0 1 2 3 4 5 6 7 8 9 10
• len(s)
• s[i], s[0]
• for i in range(n):
c = s[i]
• c in s
IMUTÁVEL
• s[ini:fim:passo]
• +, concatena
• string vazia
"" ''
len(s) = 11
Listas
(list)
Mutáveis
item in lst
lstA = lst[:]

20.3 Reuniões passadas: versão estendida

Fatias (*slices*)

Fatia (*slices*) é o mecanismo típico que usaremos para criar cópias ouu **clones** de uma lista.

O efeito da função a seguir é o mesmo que `lst[ini: fim: passo]`. Frequentemente ou quase sempre `passo == 1` e por isso escrevemos simplesmente `lst[ini:fim]`.

Para clonar a lista inteira basta escrever `lst[:]` que é o mesmo que

```
    lst[0: len(lst)]
```

```
def fatia(lst, ini, fim, passo):
    '''(list, int, int, int) -> list

    RECEBE uma lista `lst` e inteiros `ini`, `fim` e `passo`.
    RETORNA um clone da sublista de `lst` que começa na
       posição `ini` vai até `fim` (EXCLUSIVE) e agrupa
       os ítem que de passo em passo.

    O efeito é o mesmo que o fatiamento do Python:
       lst[ini: fim: passo]
    '''
    clone = []
    for i in range(ini, fim, passo):
        clone += [lst[i]]
    return clone
```

Apelido × clones: `==` e `is`

`X == Y` é `True` se (e somente se) `X` e `Y` têm o mesmo *valor*.

O operador `==` está relacionado a igualdade de valor(es).

`X is Y` é `True` se (e somente se) `X` e `Y` são apelidos para uma mesma coisa/objeto.

O operador `is` está relacionado a igualdade de identidade.

Essa explicação em Python fica assim:

```
if X == Y:
    print("X é igual a Y")
    if X is Y:
        print("Ops! Na verdade X e Y são a mesma coisa/objeto")
else:
    print("X é diferente de Y")
```

Mais um exemplo para vocês testarem

```
def main():
    lstA = [1, 'oi', True, None, 2.71828]
    # lstB é um apelido para lstA
    lstB = lstA
    print(f"lstB == lstA: {lstB == lstA}") # True
    print(f"lstB is lstA: {lstB is lstA}") # True

    # lstC é igual a lstA, mas é um clone, outra lista
    lstC = lstA[:] # ou lstA[0:len(lstA):1] ou lstA[0:len(lstA)] ou
    print(f"lstC == lstA: {lstC == lstA}") # True
    print(f"lstC is lstA: {lstC is lstA}") # False

if __name__ == "__main__":
    main()
```

Mutabilidade

Figure 2: xavier

Mutabilidade: listas são objetos, coisas **mutáveis**, ou seja *podemos alterar seus componentes*

Abaixo está um exemplo de função mutadora e de uma função que não é mutadora.

```
def main():
    # lst da main
    lst = [1, True, None, 2.71, 'x-women']
    print(lst)
    lst[0] = True
    print(lst)
    # altera lst
    mutadora(lst)
    print(lst)
    # não faz cócegas em lst da main
    nao_mutadora(lst)
```

```python
def mutadora(lst):
    '''(list) -> None
    RECEBE uma lista e ALTERA a lista que é argumento.
    Está função é MUTADORA.
    RETORNA None
    '''
    lst[len(lst)-1] = 'xavier'

def nao_mutadora(lst):
    '''(list) -> None
    RECEBE uma lista e NÃO ALTERA a lista que é argumento.
    Está função não é MUTADORA.
    RETORNA None

    Atribuição dá um apelido/nome a um objeto/coisa.
    '''
    # cria o apelido "lst da nao_mutadora"
    lst = [1, 2, 3]

if __name__ == "__main__":
    main()
```

20.4 Operador `in list`

Para verificar se um elemento `item` pertence a uma lista `lst` podemos usar o operador `in` e escrever simplesmente `item in lst`.
O valor de `item in lst` é `True` se o `item` é um elemento de `lst` e `False` em caso contrário.

```
    if item in lst:
        print(f"encontrei {item} em {lst}")
    else:
        print(f"não encontrei {item} em {lst}")
```

O efeito de `item in lst` é o mesmo que se usassemos a função de *fabricação própria* `pertence()` que está mais abaixo e escrevêssemos:

```
    if pertence(item, lst):
        print(f"encontrei {item} em {lst}")
    else:
        print(f"não encontrei {item} em {lst}")
```

A linguagem `Python` tem isso como operação nativa, basta escrever `item in lst`.

```
def main():
    lst = [1, 'oi', None, True, 3.14]
    print(f"'oi' está em lst: {pertence('oi', lst)}")
    print(f"'oi' está em lst: {'oi' in lst}")
    print(f"27 está em lst: {pertence(27, lst)}")
    print(f"27 está em lst: {27 in lst}")
    print(f"True está em lst: {pertence(27, lst)}")
    print(f"True está em lst: {True in lst}")
```

```python
def pertence(item, lst):
    '''(objeto, list) -> bool

    RECEBE um objeto/coisa item  e uma lista lst.
    RETORNA True se item é um elemento de lst, em caso
       contrário RETORNA False.

    O efeito é o mesmo que usar o operador `in` Python:
       `item in lst`
    '''
    n = len(lst)
    for i in range(n): # o mesmo que range(0, n, 1)
        if item == lst[i]:
            return True
    return False

if __name__ == "__main__":
    main()
```

20.5 Strings

Tipos nativos: `int`, `float`, `bool`, `str`, `list`, `Nonetype`

`int`, `float`, `bool` e `NoneType` são tipos de dados **primitivos**, pois seus valores *não são* compostos de partes menores. Eles não podem ser "quebrados".

Strings (`str`) e listas (`list`) são diferentes pois são compostos de partes menores.

Um **caractere** é um símbolos gráficos como letras, pontuação, espaços que são exibidos na tela. Exemplos de caracteres diferentes de letras: `' '`, `'\n'`, `'*'`, `'-'`, ...

Os componentes de uma lista podem ser **quaquer** *coisa*.

Já os componentes de strings são **somente caracteres**.

A manipulação de listas e strings é muuuito semelhante. Há, no entanto, uma diferença fundamental: strings são **imutáveis**.

Tipos formados por partes menores são chamados de coleção de tipos de dados.

Dependendo do que fizermos desejamos tratar uma coleção de tipos como uma única entidade ou desejamos acessar as suas partes.

Esta ambiguidade pode ser útil.

```
  0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| I | n | s | t | . |   | M | a | t | e | m | á | t | i | c | a |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
 -16 -15 -14 -13 -12 -11 -10 -9  -8  -7  -6  -5  -4  -3  -2  -1
```

Operador `in str`

Para verificar se uma string `s1` é **substring** de uma string `s2` podemos usar o operador `in` e escrever simplesmente `s1 in s2`.

O valor de `s1 in s2` é `True` se o `s1` é uma substring de `s2` e `False` em caso contrário.

```
if s1 in s2:
    print(f"{s1} é substring de {s2}")
else:
    print(f"{s1} não é substring de {s2}")
```

Se `c` é um caractere (= string de comprimento 1), então o efeito de `c in s` é o mesmo que se usássemos a função de *fabricação própria* `pertence()`, que está mais abaixo, e escrevêssemos:

```
if pertence(c, s):
    print(f"encontrei {c} em {s}")
else:
    print(f"não encontrei {c} em {s}")
```

A linguagem `Python` tem isso como operação nativa, basta escrever `item in lst`.

```
def main():
    s = "MAC0110 é da hora!"
    c = 'A'
    if c in s:
        print(f"encontrei {c} em {s}")
    else:
        print(f"não encontrei {c} em {s}")

    if pertence(c, s):
        print(f"encontrei {c} em {s}")
    else:
        print(f"não encontrei {c} em {s}")
```

```
def pertence(c, s):
    '''(str, str) -> bool

    RECEBE um caractere c  e uma string s.
    RETORNA True se c é um símbolo de s, em caso
       contrário RETORNA False.

    O efeito é o mesmo que usar o operador `in` Python:
       `c in s`
    '''
    n = len(s)
    for i in range(n): # o mesmo que range(0, n, 1)
        if c == s[i]:
            return True
    return False

if __name__ == "__main__":
    main()
```

20.6 Exercício - contador de vogais

Escreva um programa que lê um texto e imprime a frequência relatica de vogais.

Exemplo

```
Digite um texto: Como é bom estudar MAC0110!
Frequência das vogais = 8/27 = 0.296296

Digite um texto: Fracassei em tudo o que tentei na vida.
Frequência das vogais = 16/39 = 0.410256
```

Solução

```
VOGAIS = "aeiouáéóàãõAEIOUÁÉÓÃÕ"
def main():
    txt = input("Digite um texto: ")
    print("Texto:")
    print(f"{txt}")
    no_vogais = conte_vogais(txt)
    n = len(txt)
    print(f"Frequência das vogais = {no_vogais}/{n}\
                  = {no_vogais/n:.3g}")

# ---------------------------------------------------------
def conte_vogais(s):
    ''' (str) -> int
    RECEBE uma string s.
    RETORNA o número de vogais em s.
    '''
    cont = 0

    n = len(s)
    for i in range(n):
        c = s[i]
        if c in VOGAIS:
            cont += 1

    return cont

# ---------------------------------------------------------
if __name__ == "__main__":
    main()
```

se faltou algo, basta acrescentar

quando queremos continuar na linha de baixo

o Python percorre a string à procura de c

o mesmo funcionamento para listas

Exemplos *de luxe*

Logo passaremos a ler textos gigantescos, para isso nossos programas passarão a ler dados de arquivos.

```
Digite o nome de um arquivo: jeff.txt

We hold these truths to be self-evident:
that all men are created equal;
that they are endowed by their Creator
with certain unalienable rights;
that among these are life, liberty,
and the pursuit of happiness.

Frequência das vogais = 65/211 = 0.308057


Digite o nome de um arquivo: darci.txt

Fracassei em tudo o que tentei na vida.
Tentei alfabetizar as crianças brasileiras, não consegui.
Tentei salvar os índios, não consegui.
Tentei fazer uma universidade séria e fracassei.
Tentei fazer o Brasil desenvolver-se autonomamente
e fracassei.
Mas os fracassos são minhas vitórias.
Eu detestaria estar no lugar de quem me venceu
Darci Ribeiro

Frequência das vogais = 134/349 = 0.383954
```

Solução *de luxe*

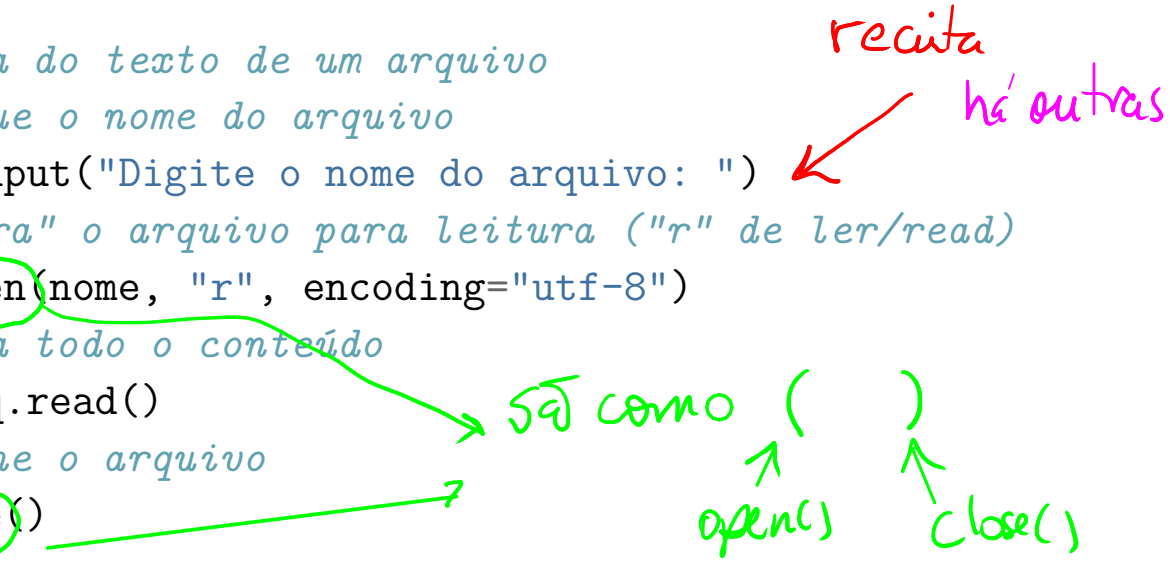
```
VOGAIS = "aeiouáéóàãõAEIOUÁÉÓÃÕ"
def main():
    # leitura do texto de um arquivo
    # 1. pegue o nome do arquivo
    nome = input("Digite o nome do arquivo: ")
    # 2. "abra" o arquivo para leitura ("r" de ler/read)
    arq = open(nome, "r", encoding="utf-8")
    # 3. leia todo o conteúdo
    txt = arq.read()
    # 4. feche o arquivo
    arq.close()

    #----------------------------
    print("Texto:")
    print(f"{txt}")
    no_vogais = conte_vogais(txt)
    n = len(txt)
    print(f"Frequência das vogais = {no_vogais}/{n} = {no_vogais/n:

# ------------------------------------------------------------
def conte_vogais(s):
    ''' (str) -> int
    RECEBE uma string s
    RETORNA o número de vogais em s.

    EXEMPLOS:

    In [2]: conte_vogais("qwrtfrhj")
    Out[2]: 0
    In [5]: conte_vogais("Como")
    Out[5]: 2
    In [3]: conte_vogais("qwrAtfrhj")
    Out[3]: 1
```



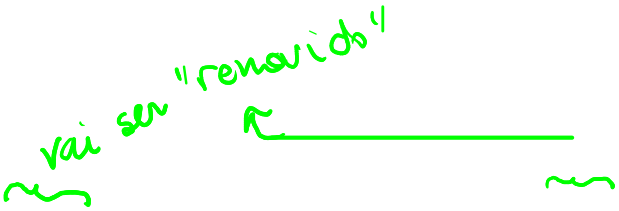
```
    In [4]: conte_vogais("MAC0110!")
    Out[4]: 1
    '''
    n = len(s)
    cont = 0
    for i in range(n):
        c = s[i]
        if c in VOGAIS:
            cont += 1
    return cont

# ---------------------------------------------------------------
if __name__ == "__main__":
    main()
```

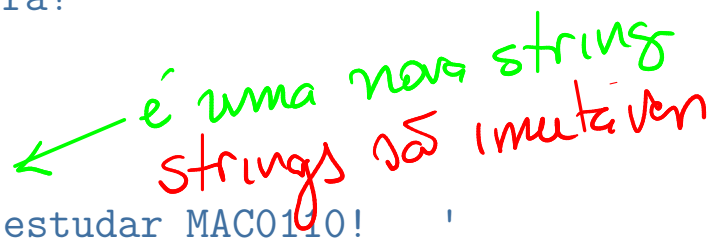
20.7 Exercício: arranca espaços

Escreva uma função `limpe()` que **recebe** uma string e **retorna** a correspondente string sem caracteres em `BRANCO` no início e no final.

Exemplos




```
In [2]: verdade = "    MAC0110!  é da hora!   "
In [3]: verdade
Out[3]: '    MAC0110!  é da hora!   '
In [4]: s = limpe(verdade)
In [5]: s
Out[5]: 'MAC0110!  é da hora!'
In [6]: declaracao = '    Amo estudar MAC0110!   '
In [7]: t = limpe(declaracao)
In [8]: t
Out[8]: 'Amo estudar MAC0110!'
```



Solução

```
BRANCO = " \n\t\r"

def limpe(s):
    '''(str) -> str

    RECEBE uma string e Retorna a correspondente string após
    o espaços em ciobranco serem removidos do início e final
    da string.

    Exemplo:


    In [8]: s = "   Python é da hora!   "
    In [90]: t = limpe(s)
    In [10: t
    Out[10]: 'Python é da hora!'
    '''
    n = len(s)
    ini = 0
    fim = n - 1

    # encontra início
    ini = 0
    while ini < n   and s[ini] in BRANCO:
        ini += 1

    # encotra fim
    fim = n-1
    while fim > ini and s[fim] in BRANCO:
        fim -= 1

    return s[ini: fim+1]
```

string vazia

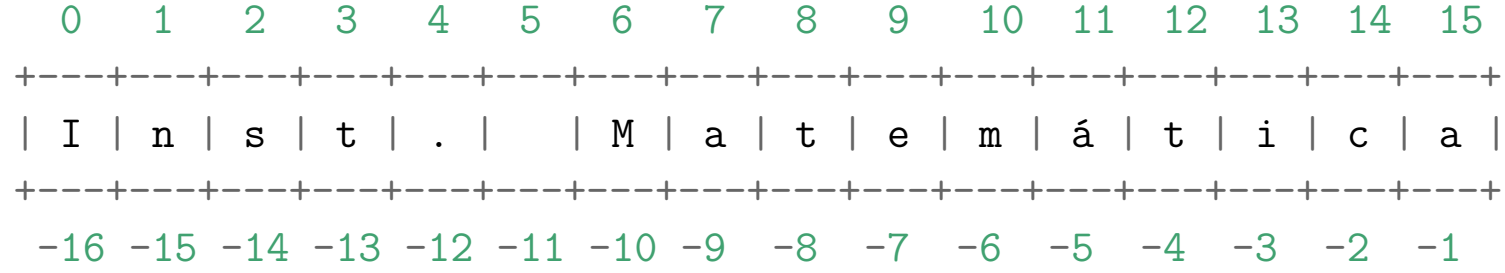
maneira de criar uma string

```
nova = ''
for i in range(ini, fim+1):
    nova += s[i]
return nova
```

o mesmo efeito que

20.8 Mais strings

```
s  = "Inst. Matemática"
```



Strings são uma sequência de caracteres delimitadas por " ou '.

Para criar uma string vazia fazemos

```
In [9]:  string_vazia = ""

In [10]: string_vazia
Out[10]: ''

In [11]:  string_vazia = ''
In [12]: string_vazia
Out[12]: ''
```

Para criar um string e um apelido para ela fazemos

```
In [13]:  s = "Como é bom estudar MAC0110!"

In [14]: s
Out[14]: 'Como é bom estudar MAC0110!'

In [15]: print(s) # o print não mostra os apóstrofos
Como é bom estudar MAC0110!
```

Strings ***são imutáveis***, não podemos alterar seus componentes

```
In [18]: s = "MAC0110"

In [19]: s[2] = 'T'    # ERRO
```

```
---------------------------------------------------------------------
TypeError                                 Traceback (most recent
<ipython-input-19-32c1c57e2e7a> in <module>
----> 1 s[2] = 'T'

TypeError: 'str' object does not support item assignment
```

Não alteramos strings, **construímos novas** strings.

```
In [23]: s = 'MAC0110'

In [24]: t = s[:2] + 'T' + s[3:]

In [25]: t
Out[25]: 'MAT0110'
```

Criar string com **f-strings** que temos usados nos `print()`

```
In [28]: n = 123
In [29]: s = f'n = {n}'

In [30]: s
Out[30]: 'n = 123'

In [31]: print(f"{s}")
n = 123

Out[37]: eu = 'Maria'

In [38]: w = f"nome = {eu}"

In [39]: w
Out[39]: 'nome = Maria'
```

Para **converter** um objeto para uma string utilizamos a função nativa `str()`.

```
In [46]: s = str(123)
In [47]: s
Out[47]: '123'
In [48]: s = str(True)

In [49]: s
Out[49]: 'True'

In [50]: s = str(None)

In [51]: s
Out[51]: 'None'

In [52]: print(s)
None

In [53]: s = str(3.1415926)

In [54]: s
Out[54]: '3.1415926'

In [55]: print(s)
3.1415926

In [56]: s = str(True)

In [57]: s
Out[57]: 'True'

In [58]: s = str([1,2,3])

In [59]: s
Out[59]: '[1, 2, 3]'
```

Percorremos strings da mesma forma que percorremos listas

```
In [58]: s = "MAC0110!"
In [58]: i = 0
In [59]: while i < len(s):
    ...:     print(f"{s[i]}")
    ...:     i + = 1
    ...:
M
A
C
0
1
1
0
!

In [60]: s = "MAC0110!"
In [61]: for i in range(len(s)):
    ...:     print(f"{s[i]}")
    ...:
M
A
C
0
1
1
0
!

In [62]: s = "MAC0110!"
In [63]: for c in s:
    ...:     print(f"{c}")
    ...:
M
```

```
A
C
0
1
1
0
!
```

A seguir está uma tabela resumindo algumas das operações sobre strings. No que segue `s` e `t` são strings e `ini`,`fim`, `passo` e `n` números inteiros.

Operação	Resultado
`t in s`	`True` se `t` é substring de `s`, senão `False`
`t not in s`	`False` se `t` é substring de `s`, senão `True`
`s + t`	a *concatenação* se `s` e `t`
`s * n` ou `n * s`	equivalente a `s` ser concatenada `n` vezes
`s[i]`	i-ésimo caractere de `s`
`s[ini:fim]`	fatia de `s` de `ini` a `fim`
`s[ini:fim:passo]`	fatia de `s` de `ini` a `fim` com passo `passo`
`len(s)`	comprimento de `s`

20.9 Mais listas ainda (`list`)

Listas são uma sequência de opjetos delimitados por `[` e `]` e separados por vírgulas.

Para criar uma lista vazia fazemos

```
In [9]:  lst_vazia = []

In [10]: lst_vazia
Out[10]: []
```

Para cria uma nova lista fazemos

```
In [13]: lst = ["M", 'A', True, None, 3.14, 123]

In [14]: lst
Out[14]: ['M', 'A', True, None, 3.14, 123]

In [15]: print(lst)
["M", 'A', True, None, 3.14, 123]
```

Listas ***são mutáveis***, *podemos* alterar seus componentes ou acrescentar alguns

```
In [18]: lst = ["M", 'A', True, None, 3.14, 123]

In [19]: lst[2] = 'T' # altera

In [20]: lst += ["x-(wo)men"] # acrescenta

In [21]: lst
Out[21]: ['M', 'A', 'T', None, 3.14, 123, 'x-(wo)men']
```

Para percorrer listas usamos os comandos `while` ou `for`.

```
In [58]: lst = [1, 2, True]
In [58]: i = 0
In [59]: while i < len(lst):
    ...:     print(f"{lst[i]}")
    ...:     i + = 1
```

```
    ...:
1
2
True

In [60]: lst = [1, 2, True]
In [61]: for i in range(len(lst)):
    ...:     print(f"{lst[i]}")
    ...:
1
2
True
```

A seguir está uma tabela resumindo algumas das operações sobre listas. No que segue `lst` é uma lista e `ini`,`fim`, `passo` e `n` números inteiros.

Operação	Resultado
`x in lst`	`True` se um item de `lst` é igual a `x`, senão `False`
`x not in lst`	`False` se um item de `lst` é igual a `x`, senão `True`
`lstA + lstB`	a *concatenação* se `lstA` e `lstB`
`lst * n` ou `n * lst`	equivalente a `lst` ser concatenada `n` vezes
`lst[i]`	i-ésimo caractere de `lst`
`lst[ini:fim]`	fatia de `lst` de `ini` a `fim`
`lst[ini:fim:passo]`	fatia de `lst` de `ini` a `fim` com passo `passo`
`len(lst)`	comprimento de `lst`